CLAIMS

1. A computer system providing an object-based environment, said computer system including storage, a contiguous linear portion of which is logically divided into first and second heaps located at opposite ends of the storage portion, with any gap between the two heaps representing an unallocated region of storage, wherein references are permitted from objects on the first heap to objects on the second heap and vice versa, said system comprising:

a garbage collector for operating across both heaps to remove objects that are no longer live;

means for expanding the first heap into said unallocated region according to a first expansion policy; and

means for expanding the second heap into said unallocated region according to a second expansion policy.

2. The computer system of claim 1, wherein said computer system supports a transaction processing environment, and said first heap is used for storing objects that are deleted at the end of the current transaction, and said second heap is used for storing objects that persist from one transaction to another.

3. The computer system of claim 2, wherein the first heap is reset to the same predetermined initial size at the start of each transaction.

4. The computer system of claim 3, wherein the system returns an error condition if the second heap has expanded such that it is not possible to reset the first heap to its predetermined initial size.

5. The computer system of claim 3, wherein a midpoint is defined halfway between the first heap and second heap, when they each have their initial size.

6. The computer system of claim 5, wherein the first expansion policy is always to expand into said unallocated region in order to satisfy a storage request.

7. The computer system of claim 6, in which the rate of expansion of the first heap into the unallocated region is slower once the first heap has passed said midpoint.

8. The computer system of claim 5, wherein the second expansion policy is to expand into said unallocated region in order to satisfy a storage request until said midpoint is reached, whereupon said system preferentially performs a garbage collection to satisfy said request.

9. The computer system of claim 8, wherein the second expansion policy further includes trying to shrink the first heap to allow room to expand said second heap in order to satisfy a storage request.

10. The computer system of claim 1, wherein said garbage collector performs a compact operation after a garbage collection of the first and second heaps, said compact operation being performed in response to a first set of

criteria relevant to the first heap, and a second set of criteria relevant to the second heap.

11. The computer system of claim 10, where said second set of criteria are more sensitive to fragementation than the first set of criteria.

12. The computer system of claim 10, further including means for shrinking the first and second heaps after compaction, and returning released storage to said unallocated region.

13. The computer system of claim 1, further including a bit array, having one bit for each possible object location in said portion of storage, said bit indicating whether or not there is an object currently stored at the corresponding object location.

14. A method of operating a computer system providing an object-based environment, said computer system including storage, a contiguous linear portion of which is logically divided into first and second heaps located at opposite ends of the storage portion, with any gap between the two heaps representing an unallocated region of storage, wherein references are permitted from objects on the first heap to objects on the second heap and vice versa, said method comprising the steps of:

operating a garbage collector across both heaps to remove objects that are no longer live;

expanding the first heap into said unallocated region according to a first expansion policy; and

for expanding the second heap into said unallocated region according to a second expansion policy.

15. The method of claim 14, wherein said computer system supports a transaction processing environment, and said first heap is used for storing objects that are deleted at the end of the current transaction, and said second heap is used for storing objects that persist from one transaction to another.

16. The method of claim 15, wherein the first heap is reset to the same predetermined initial size at the start of each transaction.

17. The method of claim 16, wherein the system returns an error condition if the second heap has expanded such that it is not possible to reset the first heap to its predetermined initial size.

18. The method of claim 16, wherein a midpoint is defined halfway between the first heap and second heap, when they each have their initial size.

19. The method of claim 18, wherein the first expansion policy is always to expand into said unallocated region in order to satisfy a storage request.

20. The method of claim 19, in which the rate of expansion of the first heap into the unallocated region is slower once the first heap has passed said midpoint.

21. The method of claim 18, wherein the second expansion policy is to expand into said unallocated region in order

to satisfy a storage request until said midpoint is reached, whereupon said system preferentially performs a garbage collection to satisfy said request.

22. The method of claim 21, wherein the second expansion policy further includes trying to shrink the first heap to allow room to expand said second heap in order to satisfy a storage request.

23. The method of claim 14, wherein said garbage collector performs a compact operation after a garbage collection of the first and second heaps, said compact operation being performed in response to a first set of criteria relevant to the first heap, and a second set of criteria relevant to the second heap.

24. The method of claim 23, where said second set of criteria are more sensitive to fragementation than the first set of criteria.

25. The method of claim 23, further including the step of shrinking the first and second heaps after compaction, and returning released storage to said unallocated region.

26. The method of claim 14, further including the step of providing a bit array, having one bit for each possible object location in said portion of storage, said bit indicating whether or not there is an object currently stored at the corresponding object location.

27. A computer program product comprising program instructions recorded on a storage medium readable by a computer system, said computer system providing an object-based environment and including storage, a contiguous linear portion of which is logically divided into first and second heaps located at opposite ends of the storage portion, with any gap between the two heaps representing an unallocated region of storage, wherein references are permitted from objects on the first heap to objects on the second heap and vice versa, said program instructions causing said computer system to perform the steps of:

operating a garbage collector across both heaps to remove objects that are no longer live;

expanding the first heap into said unallocated region according to a first expansion policy; and

for expanding the second heap into said unallocated region according to a second expansion policy.

28. The computer program product of claim 27, wherein said computer system supports a transaction processing environment, and said first heap is used for storing objects that are deleted at the end of the current transaction, and said second heap is used for storing objects that persist from one transaction to another.

29. The computer program product of claim 28, wherein the first heap is reset to the same predetermined initial size at the start of each transaction.

30. The computer program product of claim 29, wherein the system returns an error condition if the second heap

has expanded such that it is not possible to reset the first heap to its predetermined initial size.

31. The computer program product of claim 29, wherein a midpoint is defined halfway between the first heap and second heap, when they each have their initial size.

32. The computer program product of claim 31, wherein the first expansion policy is always to expand into said unallocated region in order to satisfy a storage request.

33. The computer program product of claim 32, in which the rate of expansion of the first heap into the unallocated region is slower once the first heap has passed said midpoint.

34. The computer program product of claim 31, wherein the second expansion policy is to expand into said unallocated region in order to satisfy a storage request until said midpoint is reached, whereupon said system preferentially performs a garbage collection to satisfy said request.

35. The computer program product of claim 34, wherein the second expansion policy further includes trying to shrink the first heap to allow room to expand said second heap in order to satisfy a storage request.

36. The computer program product of claim 27, wherein said garbage collector performs a compact operation after a garbage collection of the first and second heaps, said compact operation being performed in response to a first

set of criteria relevant to the first heap, and a second set of criteria relevant to the second heap.

37. The computer program product of claim 36, where said second set of criteria are more sensitive to fragementation than the first set of criteria.

38. The computer program product of claim 36, wherein said program instructions further cause said computer system to perform the step of shrinking the first and second heaps after compaction, and returning released storage to said unallocated region.

39. The computer program product of claim 27, wherein said program instructions further cause said computer system to perform the step of providing a bit array, having one bit for each possible object location in said portion of storage, said bit indicating whether or not there is an object currently stored at the corresponding object location.